Improved Streaming Quotient Filter: A Duplicate Detection Approach for Data Streams

Shiwei Che, Wu Yang, and Wei Wang
Information Securityresearch Center, Harbin Engineering University, China

Abstract: *The unprecedented development and popularization of the Internet, combined with the emergence of a variety of modern applications, such as search engines, online transactions, climate warning systems and so on, enables the worldwide storage of data to grow unprecedented. Efficient storage, management and processing of such huge amounts of data has become an important academic research topic. The detection and removal of duplicate and redundant data from such multi-trillion data, while ensuring resource and computational efficiency, has constituted a challenging area of research.Because of the fact that all the data of potentially unbounded data streams can not be stored, and the need to delete duplicated data as accurately as possible, intelligent approximate duplicate data detection algorithms are urgently required. Many well-known methods based on the bitmap structure, Bloom Filter and its variants are listed in the literature. In this paper, we propose a new data structure, Improved Streaming Quotient Filter (ISQF), to efficiently detect and remove duplicate data in a data stream. ISQF intelligently stores the signatures of elements in a data stream, while using an eviction strategy to provide near zero error rates. We show that ISQF achieves near optimal performance with fairly low memory requirements, making it an ideal and efficient method for repeated data detection. It has a very low error rate. Empirically, we compared ISQF with some existing methods (especially Steaming Quotient Filter (SQF)). The results show that our proposed method outperforms theexisting methods in terms of memory usage and accuracy.We also discuss the parallel implementation of ISQF.*

Keywords: *Bloom filters, Computer Network, Data stream, Duplicate detection, False positive rates.*

Received November 30, 2017; accepted July 21, 2019
https://doi.org/10.34028/iajit/17/5/10

1. Introduction

At present, in many areas there have been increasing amounts in data, such as social network, information retrieval, video surveillance, finance, energy industries, and so on, data intensive computing method has become a main research field of industry and research community. Managing and processing such large amounts of data is a challenging task, and duplication or redundancy of data further exacerbates the difficulty of the task, leading to a waste of valuable storage and computing resources. Removing duplicate data from such data sources can improve application perfomance. This paper addresses the problem of detecting and dleting duplicate data in a data stream environment to mitigate the computational pressure of processing such data sources. In the case that an element in the data stream appears before, it is considered to be a duplicate or a redundant element. The problem of removing such elements is called repeated data removal problem.

In traditional query processing and data stream management system [3], deleting duplicate elements is an important operation. To this end, researchers have proposed several classic algorithms [20], such as aproximate frequency moments [1], element classifiction [23], correlated aggregate queries [21], etc., The real-time nature of the de-duplication problem demands efficient in-memory algorithms, but theinability to store the whole stream (possibly infnite) makes exact duplicate detection infeasible in streaming scenarios.Thus, in most cases, a fast method with tolerable error rate is acceptable at the expense of accuracy. In this paper, we propose a new efficient approximation algorithm to solve the repeated data detection problem in data streams.

In large telecommunication networks where Call Data Records (CDR) aregenerated, errors in the CDR generation mechanisms may result in repeated data generation.To ensure computational efficiency, real time periodic deletion of duplicate data should be performed on a data set containing approximately 5 bilion multi-dimensional records before storing the CDR into a permanent storage device. In this case, the classic database access solution is extremely slow, while the classic Bloom Filter [6] approach is extremly resource intensive.

Network monitoring and accounting provide an analysis of network users and their usage patterns, which are widely used in recommender systems and personalized web search. The classification of users into new or existing ones and updating their description information is an interesting application fordeduplication. Deduplication in the click stream also helps prevent fraud in the area of Web advertising and prevents site publishers from gaining more profits from the advertiser through fake clicks. The detection of duplicate user IDentifier (ID), Internet Protocol (IP), etc., encourages to reduce such fraudulent activities.

Search engines regularly crawl web pages to update their corpus of extracted Uniform Resource Locators (URLs). Given a newly extracted URLs table, the search engine must scan its archive database todetermine whether the URL is already in the library. If not, deposit it. With the exponential growth of web pages, data deduplication becomes indispensable in such scenarios.Imprecise duplicate detection may lead to have identical URL in the library (False Negative, (FN)), thereby reducing the search engine performance, or may lead to a new page to be ignored (False Positive, (FP)) making the corpus of stale.

As can be seen from the above examples, the actual application has a strong need for repetitive data deletion algorithms that can run in memory, work in real time, and have low error rates.

In this paper, we propose an Improved Streaming Quotient Filter (ISQF), which enhances the Streaming Quotient Filter (SQF) structure. On top of this structure, we propose a new algorithm for detecting duplicate data in data streams. Then, we discuss the implementation of ISQF on a parallel architecture. We experimentally analyzed the performance of ISQF. The obtained results show that the error rate of ISQF is very low, and for large streams, the error rate is close to zero, which makes the ISQF much better than the existing methods. We also discuss the parameter setting problem of the algorithm, and give the empirical results on three large scale synthetic data setsand weshow the efficiency of ISQF in terms of convergence, low error rate, and memory requirements.

The rest of this paper is organized as follows. Section 2 gives a precise definition of the problem as well as a background study of the existing structures and methods. Section 3 gives a brief introduction to the main comparison structure SQF of the structure ISQF proposed in this paper. Section 4 gives a detailed description of the structure ISQF and the corresponding algorithm proposed in this paper for repeated data detection in data streams, and analyzes the performance of the algorithm. Section 5 presents the implementation of ISQF on parallel architectures. The experimental evaluation results of ISQF are given in section 6. Section 7 finally concludes the paper.

2. Preliminaries and Related Work

2.1. Problem Definition

A data stream is a sequence of elements, $S=e_1, e_2 \ldots e_N$, the length of the data stream N may be infinite. We assume that the stream elements are obtained uniformly from a finite alphabet set Γ with cardinality U, namely: $|\Gamma|=U$. Then, each element of the stream is converted to a number using the hashing method or the fingerprint generation method. The deduplication problem can be described as: given a data stream S, and a certain number of memory M, reporting whether each element e_i in S already appeared in $e_1, e_2, \ldots, e_{i-1}$ or not. Since storing all stream data is infeasible, approximate deduplication algorithms are needed to minimize the error.

2.2. Pairwise Independent Hash Functions

Here, we only briefly introduce the definition of pairwise independent hash functions.

A family of functions $H = \{h|h(\cdot) \to [1, w]\}$ is called the family of pairwise independent hash functions if for two different hash keys x_i, x_j, and $k, l \in [1, w]$,

$$\Pr_{h \leftarrow H}\left[h(x_i) = k \wedge h(x_j) = l\right] = 1/w^2 \quad (1)$$

Intuitively, in order to reduce the probability of hash collisions, the hash functions used should be pairwise independent when multiple hash functions are used. For more details, please refer to [12].

2.3. Related Work

The Navie method for detecting duplicate data in a data stream involves database and document query, or pair-wise string comparison, which is extremely slow, and disk access can corrupt the real-time nature of the problem. The simple cache and buffer methods include using stream elements to populate a fixed size buffer, and to check the presence of each new element in the buffer. When the buffer is full, a policy is used to remove an element from the buffer to store the new element. Several eviction strategies are discussed in [20]. However, experiments show that the performance of the buffer technology greatly depends on the expulsion strategy adopted and the stream behavior.

The problem of bit shaving to address fraudulent advertiser traffic was investigated in [29]. In [10, 11, 26], an approximate duplicate data detection method for search engines and Web applications is proposed. [2, 18] uses file level hashing in storage systems to detect duplicates, but it suffers from low compression rates. In [22], secure hash techniques for fixed size data blocks are studied.

In order to solve the computational challenge of detecting duplicate data in data streams, Bloom Filters is commonly used in applications such as [7, 9, 14, 15, 25, 27, 31]. Bloom Filter provides a spatially efficient probabilistic snapshot structure for the membership queries in sets. A Bloom Filter is a bit vector consisting of m components with the initial values set to 0. The classic Bloom Filter method involves the comparison of the k selected bits of the vector obtained by the k hash functions to determine if the elements are repeated. Inserting an element e_i also involves setting the k components of the Bloom Filter computed by the k independent hash functions $h_1(e_i), h_2(e_i), \ldots, h_k(e_i)$. Hoever, the memory and computational efficiency achieved are at the expense of a small error rate. Researchers have also proposed disk based Bloom Filters, but the overall performance of the structure has been declined.

In order to support the continuous insertion and deletion of elements of the bloom filter structure, [18] introduces counting Bloom Filters. This method replaces the bits in the filter with a small counter, which maintains the number of elements that hash to a particular location. There are some other variants of the bloom filter model which include compressed Bloom Filter [28], spacecode Bloom Filters [25], Decaying bloom filter [30], and spectral bloom filters [12], etc.,

Bloom Filters has also been applied to network related applications, such as heavy flows for stochastic fair bluequeue management [19] to assist routing, packet classification [4], state management of each flow, and longest prefix matching [14]. [24] Extendsbloomjoin for distributed joins to minimize the network usage during database statistic query execution. For multicore applications, a parallel version of Bloom Filters has been proposed by [26]. An interesting Bloom Filter structure, Stable Bloom Filter (SBF), proposed by [13] provides a guarantee regarding the performance of the structure. It removes elements from the structure continuously and provides a constant upper bound on False Positive Rate (FPR) and False Negative Rate (FNR). This stable performance provides a powerful guarantee for the real-time efficiency of deduplication applications. However, SBF has a higher rate of false positive; theoretically, it can achieve convergence only on infinitely long streams. In order to overcome the shortcomings of SBF, [16] prposed Reservoir Sampling based Bloom Filter (RSBF), which is a new combination of reservoir sampling technology with Bloom Filter structure.

The problem of high FNR is primarily caused by the deletion of a bit in the Bloom Filter that may cause the logical deletion of more than one element, which means, deleting all the elements that map to this bit. Quotient filter [5] uses quotienting technology to eliinate this problem, allowing Bloom Filter to be "dletion-friendly". However, it does not support data stream queries.

In [17], SQF was proposed, which is an improved version of quotient filter. Compared with the method mentioned above, the accuracy is greatly improved, the error rate is highly reduced, and only a small amount of memory is used. Therefore, we use it as the main comparison method with the one described in this paper. In order to make a good contribution to our approach, we will briefly introduce it in the next section.

In this article, we propose the ISQF for deduplication problems. Compared with existing structures, it provides near zero error rates using a small amount of memory. As far as we know, ISQF provides the lowest error rate compared with the previous methods, and the memory requirement is very low. We also discussed how ISQF works on a parallel architecture. We believe that using parallel and distributed settings, ISQF can efficiently manage petabytes of data with relatively low memory (on the order of hundreds of gigabytes).

3. Streaming Quotient Filter

To better illustrate ISQF, in this section, we briefly introduce SQF [17].

Dutta *et al.* [17] Assumes that each element of the stream S is uniformly drawn from a finite universe Γ, the cardinality of Γ is U and it is hashed to a p-bit fingerprint using Rabin's method [21]. This assumption is also applicable to our approach presented in this paper. The fingerprint of the element is the input of the SQF structure F. conceptually $F = h(S) = \{h(e) | e \in S\}$, where $h: \Gamma \rightarrow \{0,1,\ldots,2^p - 1\}$.

SQF stores the signature formed by the input element fingerprint in a hash table T, and the number of rows in T is R=2^q (q<p). In this method, the input fingerprint of an element e, f_e, is divided into its r least-significant bits $f_e^r = f_e \bmod 2^r$(remainder), and its q=p-r most-significantbits $f_e^q = \lfloor f_e/2^r \rfloor$ (quotient). Each row of the hash table T is further divided into k buckets, each storing the signature of an element. The signature σ_e of an element e consists of $f_e^{r'}$ followed by O_e. Among them, O_e is the number of 1 in f_e^r, and $f_e^{r'}$ is a number formed by selecting r' bits from f_e^r. $f_e^{r'}$ is called a reduced remainder. Figure 1 describes the structure of the SQF [17].

When the signature σ_e of a new element e in the data stream arrives, SQF stores the signature σ_e in the bucket of row f_e^q in hash table T. Therefore, the membership query of a data stream element e only involved checking the presence of the signature of the element e in the buckets of the corresponding row (row f_e^q) of the hash table, which provides a concise and efficient algorithm for the detection of duplicate data.

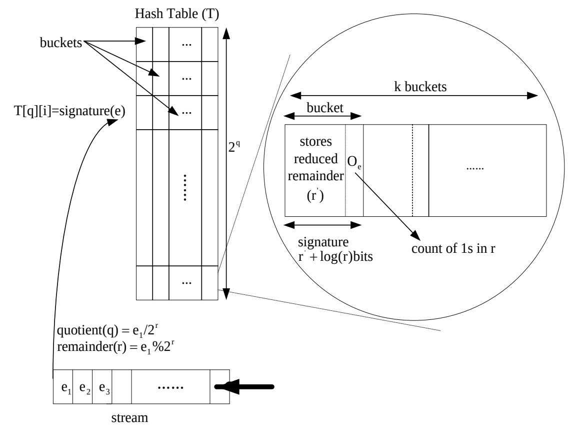


Figure 1. The structure of SQF.

If two fingerprints, f and f', are mapped to the same row of T, that is, there is the same quotient ($f^q = f'^q$), as in [5], we call it a soft collision. Since each row of T contains only k buckets, when more than k distinct elements mapped to the same row of T, we call it a hard collision, as in [17]. When a hard collision occurs, the algorithm uniformly and randomly chooses an element signature out of k stored signatures in the buckets, and it removes the signature and stores the new signature. For an example of using SQF, please refer to the example in section 4.

When the hard collision occurs, 0after deleting the signature, the same signature of the same element in data flow appears again, because the same signature is deleted earlier, the algorithm will not be able to detect the later elements that are repeated, which caused a large false negative existence, reduces the accuracy of the algorithm. Simply increasing k to expand the bucket number of each row of the hash table T to reduce the number of missing reports will also encounter a problem of a too large average lookup length (k/2, which is discussed later in this article). The above two questions are exactly the problems that ISQF structure in this article should solve.

4. Improved Streaming Quotient Filter ISQF)

4.1. Principle and Algorithm

In order to solve the two shortcomings of the SQF proposed at the end of the last section, we propose ISQF. Its structure is illustrated in Figure 2.

ISQF consists of m hash tables. Each hash table is equivalent to a SQF. Every table contains 2^q entries, and each of them containing k buckets and each bucket stores the signature of a data stream element. The hash functions of this m hash tables are m pairwise independent hash functions $h_i(\cdot)\rightarrow[0,2^q-1], i=1,2\ldots,m$.

When the fingerprint f_e of a new data stream element e arrives, the algorithm first calculates $f_e^q = \lfloor f_e/2^r \rfloor$, $f_e^r = f_e \bmod 2^r$, O_e indicates the number of bits set to 1 in f_e^r. The $r'(<r)$ bits are selected from the f_e^r by a function $\Omega : \mathbb{R}^r \rightarrow \mathbb{R}^{r'}$, and the reduced remainder $f_e^{r'}$ is formed by this r' bit. The signature σ_e of the element e is made up of $f_e^{r'}$ followed by O_e. Then, iterate through the following operations on the m hash tables: in the ith iteration, calculate the hash function $h_i(f_e^q)$, check whether there is the same signature with σ_e in the k buckets of the entry $h_i(f_e^q)$ of the ith hash table T_i, if:

1. having the same signature, then the algorithm reports that e is a repeating element and terminates the iteration.
2. otherwise, σ_e is stored in an empty bucket of the entry and continues iteration.
3. if the entry is full and no empty bucket exists, then selects one of the k buckets uniformly and randomly, deletes the existing signature, stores the new signature, and continues the iteration. The following example illustrates the above implementation of the algorithm.

- *Example*: assuming that the fingerprint $f_e(p=8)$ of an element e is $(10011011)_2$, and r=4. Therefore, q=p-r=4, each hash table of ISQF contains $R=2^q = 16$ entries. Suppose there are m=3 hash tables in the ISQF structure, and each entry in the hash table contains k=1 bucket. Then, $f_e^q = (1001)_2$, $f_e^r = (1011)_2$. Let Ω be a simple function selecting $r' = 2$ most-significant bits

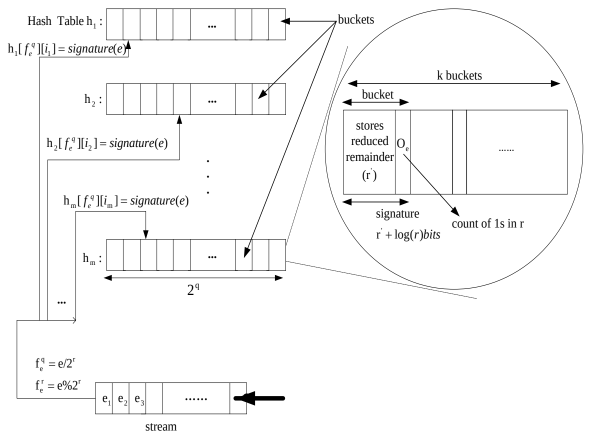


Figure 2. The structure of ISQF. of f_e^r.

Thus, the reduced remainder $f_e^{r'}$ is $(10)_2$, the number of 1 in f_e^r is $O_e=3=(11)_2$, and the signature of element e is σ_e=1011, generated by $f_e^{r'}$ followed by O_e. Assume that $h_1(f_e^q) = h_1(9) = 15$, $h_2(f_e^q) = h_2(9) = 23$, $h_3(f_e^q) = h_3(9) = 8$, and the corresponding entries of the three hash table in which are not full, for the fifteenth entry of the first hash table (the entry count from 0), the twenty-third entry of the second hash table and the eighth entry of the third hash table, the signature σ_e is stored in anempty bucket.

The following algorithm 1 describes the pseudo code of ISQF.

Algorithm 1: ISQF(S)

```
Require: Stream(S), Number of bits in the fingerprint of elements (p), the number of memory bits available (M), the bit selection function (Ω), the number of hash tables (m)
Ensure: Detecting duplicate and unique elements in S with low error rates
Set parameters: the number of bits in the remainder ($r(< p)$), the number of buckets in each entry of the hash table (k), and the size of each bucket in bits ($s_b$)
2: Calculating bits number of quotient, $q \leftarrow p - r$. Create m hash tables, each with $R \leftarrow 2^q$ entries
3: Every entry of the m hash tables is further divided into k buckets, the size of each is $s_b$ bits. Initially, the m hash tables are all empty
```

```
4: sp=1
5: e=Current element S[sp] of stream S
6: Result←DISTINCT
7: Let $f_e$ is the p bits fingerprint of e
8: Calculate quotient of $f_e$, $f_e^q \leftarrow \lfloor f_e/2^r \rfloor$
9: Calculate remainder of $f_e$, $f_e^r \leftarrow f_e\%2^r$
10: $O_e$ ←number of one in $f_e^r$
11: Select the $r'$(<r) bits positions using the function Ω
12: The reduced remainder of e, $f_e^{r'}$ ←the numbe
formed by the selected $r'$ bits
13: The signature $\sigma_e$ of e is obtained by $f_e^{r'}$ and $O_e$
14: i=1
15: Let $T_i$ be the ith table of the m hash tables
16: n=$h_i(f_e^q)$
17: j=1
18: Let $b_j$ is the jth bucket of entry n of T($T_i[n]$)
19: If Signature at $b_j$=$\sigma_e$, then Result=DUPLICATE and go to
21
20: If j<k, then j=j+1 and go to 18
21: If Result= DUPLICATE, then go to 23
22: If i<m, then i=i+1 and go to 15
23: If Result=DISTINCT, then i=1; otherwise, go to 30
24: Let $T_i$ be the ith table of the m hash tables
25: n=$h_i(f_e^q)$
26: Let $b_{empty}$ be an empty bucket at the entry n of $T_i(T_i[n])$
27: If $b_{empty}$ does not exist, i.e., there is no empty
bucket in $T_i[n]$, then $b_{empty}$ ←select a bucket uniformly and
randomly from $T_i[n]$
28: Storing $\sigma_e$ in bucket $b_{empty}$
29: If i<m, then i=i+1 and go to 24
30: If sp< the length of stream S, then sp=sp+1 and go to 5;
otherwise, stop the algorithm and output Result
```

4.2. Analysis

The pervious section shows that ISQF uses m (>1) hash tables to store the signatures of data stream elements, which greatly reduces the false negative rate compared with SQF. SQF has only one hash table, each table entry has k buckets. When a hard collision occurs, it randomly selects a signature to remove in order to make a storage space for storing new elements. When the same element in the stream arrives again, SQF cannot detect that it is a repeating element because it has been deleted from SQF, resulting in a failure to report; while ISQF uses m (>1) hash tables to store signatures, when one or more hash tables have a hard collision and cannot detect duplicate elements, as long as a hash table does not have a hard collision or it occurs, but the deleted signature is not the duplicate element signature, the ISQF can detects duplicate elements and it will not be missed. Since there are multiple hash tables, this is exactly a high probability event.

Then why not extend the storage space of a hash table of SQF directly to reduce the hard collision but rather build multiple hash tables? This problem involves the concept of the Average Search Length (ASL) of data structure discipline. Suppose there are 8 hash tables in ISQF, each table entry has 120 buckets. Then, with the same storage space, each table entry for SQF can have 960 buckets. From the definition of SQF, the signatures in SQF are uniformly distributed. Thus, under the above conditions, the average search length for finding a repeating element in SQF is ASL_{SQF}=480. Since ISQF's m hash functions are pairwise independent hash functions, the signatures in each table of ISQF are also uniformly distributed. Then, under the above conditions, assume that 3 tables in 8 hash tables still have signatures of repeating elements, the average search length for finding a repeating element in ISQF is ASL_{ISQF}=320. Which is significantly lower compared with SQF. And, as the number of hash tables m and the number of buckets k in each table entry increases, this decrease in ASL becomes more pronounced.

We now consider the memory space required by ISQF. From [8] we can see, the number of memory bits required for SQF is

$$M = k \cdot 2^q \cdot (r' + \log r) \tag{2}$$

The meaning of k, q, r and r' in the formula are the same as above. Consider the examples in section 4.1, we used p=8, q=4, r= 4, k=1, r'=2, M=64 bit [8]. That is, it uses 8 bytes to store the signatures of 16 different elements, which are very low memory requirements. Considering that our ISQF just transforms a hash table of SQF into m (>1), this memory demand is still low and is supported by most systems.

5. Parallel Implementation

In this section, we describe the implementation of ISQF on a parallel architecture. This enables ISQF to efficiently handle deduplication of petabytes of data using low memory space (hundreds of gigabytes).

Suppose that a parallel environment consists of P processors, and the input stream is partitioned into blocks of C elements. Each element in a block b_i is evenly assigned to the P processor, along with its position pos in b_i. Hence, each processor receives α=C/P elements of block b_i. Every hash table of the stored signatures is evenly distributed over the P processors, each node contains R/P contiguous entries of each hash table (R is the length of each hash table). Since there are m hash tables, each node contains a total of m * R/P hash table entries, and each hash table's entries are contiguous. Each node holds the hash functions of the m hash tables to calculate the m hash values of the inputed element's fingerprint. Each processor runs two parallel threads, p_1 and p_2. This can easily be extended to multiple threads.

For the C elements of block b_i, each node N_S receives the element fingerprints e_{sa}, e_{sa+1},…, $e_{(s+1)\alpha-1}$ $s\in[0, P\text{-}1]$. Suppose that the m hash tables are $H_{d_1}, H_{d_2}, \dots, H_{d_m}$, respectively. The thread p_1^s of the node N_S calculates the m hash values of the inputed element fingerprint e_k^s. According to the calculated hash values, N_S passes e_k^s, its corresponding hash value and position pos to the node $N_{d_1}, N_{d_2}, \ldots, N_{d_m}$ that contain the corresponding entries of the hash tables. The second thread $p_2^{d_i}$ of node $N_{d_i}(i = 1,2,\ldots,m)$ calculates the signature of the e_k^s and checks whether the fragment H_{d_i} of the hash table which it holds contains the signature. As long as there is a thread $p_2^{d_i}(i = 1,2,\ldots,m)$ found in the local stored hash table entries containing the signature, the algorithm reports the corresponding element as a repeating element. Otherwise, $p_2^{d_i}$(i=1, 2,…, m) stores the signature in the hash table H_{d_i}. However, if the membership query is the same as before, the accuracy of the algorithm may be reduced. For example, considering the case of m=1, C=P=3, the elements of a block to be {u, v, v*}, wherev= v*, v* arrives after v, and v has never appeared before in the stream. Therefore, v should be reported as distinct element, and v* should be reported as repeating element. However, if the node N_3 stores a fragment that v and v* mapped to the hash table T, and N_3 handles v* before v, then the algorithm will report that v*is the distinct element, and v is the repeating element.

To solve this problem, each bucket in the hash table should contain an additional field which is the position in the block of the stored element in the bucket, represented by the variable pos. Consider the example of single hash table, where the node N_d receives the fingerprint e_k^s and checks the presence of e_k^s's signature in the local hash table. If the signature does not exist, N_d stores it in a bucket B and sets the pos field to $pos_{e_k^s}$. Thus, when a subsequent fingerprint $E=e_k^s$ arrives, the N_d checks the pos field of the bucket B. If $pos_E > B_{pos}$, then E is located behind e_k^s in the stream, so N_d correctly informs that the element corresponding to E is the repeating element. If $pos_E < B_{pos}$, indicateing that E is before e_k^s in the stream, but it is processed after e_k^s. Therefore, N_d notifies the element corresponding to e_k^s as the repeating element and updates B_{pos} to pos_E. After the entire block has been processed, the element of the signature stored in the bucket is reported as the distinct element. Then, the next fingerprint block of elements is processed.

For the above example, now we assume that N_3 handles v* first. Because its signature is not in the hash table T, N_3 stores the signature in the bucket B and sets B_{pos}=3, resulting from that it is the third element in the block. When v is processed later, the signature of v is found in T, since the same element v* is first processed. Now, N_3 checks the pos field of B, because pos_v=2< B_{pos}, so N_3 correctly infers that v comes before v* in the stream. Therefore, N_3 informs that the third element is duplicated and updates B_{pos} to 2. After the elements in the block are processed, the algorithm notifies that the second element is the unique element.

6. Experiment

In this section, we experimentally evaluate the performance of ISQF by comparing it with the state of art methods. SBF, RSBF and SQF are the three most advanced structures in this field. Using three synthetic data sets, we compared the error rates of our proposed ISQF structure and the three structures mentioned above.

The synthetic data sets are generated using uniform distribution. In order to capture a wide variety of stream scenarios, the percentage of unique elements in each synthetic data set is different.

First we discuss the settings for the parameters r, r′, k, and m. Then, using these settings, based on the changes in the following scenarios, we conduct experiments to capture variations of error rates of various methods:

- Number of input records.
- Percentage of different elements.
- Memory requirements.

6.1. Setting of Parameters

Fan *et al*. [18] Shows that for a single hash table, the optimal settings for the parameters r, r′, and k are r=2, k=4, r′=r/2=1. In this paper, we use these settings. In the following section, we discuss the impact of the parameter m setting on algorithmic error rates. Table 1 and Figure 3 describes the change in the error rate of the algorithm on the third synthetic data set as the parameter m changes:

Table 1. The impact of m values on algorithmic error rates.

m	**Error rate(%)**
1	0.0035
2	0.0021
3	0.0013
4	0.00029
5	0.00021
6	0.00007
7	0.00006
8	0.00001

As can be seen from Table 1 and Figure 3, with the increasing of m value, the error rate of the algorithm is significantly decreased and the accuracy is greatly increased. By adjusting the value of m, the memory size is regulated accordingly, this paper demonstrates the error rate of the algorithm using different memory space sizes on three synthetic datasets.

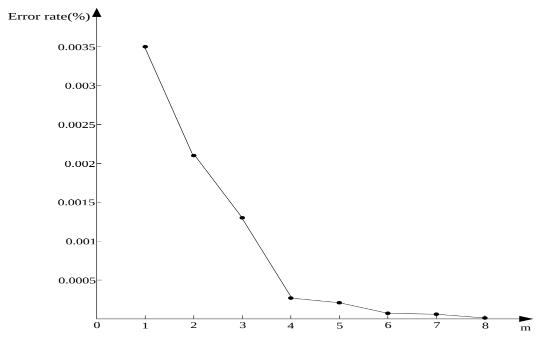


Figure 3. The Impact of M values on algorithmic error rates.

6.2. Synthetic Datasets

In this part, we demonstrate the performance of Improved ISQF using three large data sets generated by uniformly distributed random data. To simulate the real stream scenes, we use a different percentage of the unique elements in the three synthetic data sets. Table 2 indicates the performance of ISQF and other algorithms using different memory sizes on the data sets.

Table 2. Performance comparisons on composite data sets.

Datasets (%Distinct)	Memory	Error rate (%)			
		SBF	RSBF	SQF	ISQF
1 billion (15%)	64MB	2.92	4.29	0.0042	0.0018
	128MB	1.26	1.61	0.0006	0.00027
	512MB	0.13	0.14	0.0001	0.00002
695 million (60%)	128MB	4.31	6.73	0.003	0.0019
	256MB	2.09	2.77	0.001	0.00012
	512MB	0.82	0.92	0.0003	0.00004
100 million (90%)	32MB	4.13	6.04	0.0032	0.0021
	64MB	2.05	3.17	0.0007	0.00029
	128MB	0.64	1.12	0.0001	0.00001

It can be seen from Table 2, for a data set containing 1 billion records with 15% unique elements, using 64MB memory, ISQF achieves a 0.0018% error rate, while the current state-of-the-art approach, SQF, has a 0.0042% error rate. As a result, ISQF achieves a performance improvement of nearly 24 ×.

For a data set containing 695 million records with 60% unique elements, using 256MB memory, ISQF achieves a 0.00012% error rate, while SQF has a 0.001% error rate. As a result, ISQF achieves a performance improvement of nearly 10×.

The third synthetic data set contain 100 million records, with 90% unique elements (as shown in Table 2). It can be seen from Table 2, on the third data set, ISQF has a similar performance to the above analysis. When allocating 128MB memory, the error rate of ISQF is 0.00001%, while the error rate of SQF is 0.0001%, the error rate of SBF is 0.64%, and the error rate of RSBF is 1.12%. Therefore, ISQF is far superior to SQF, SBF and RSBF.

7. Conclusions

In a stream scenario, taking into account numerous data from various applications, real-time deduplication in memory poses a challenging problem. In this paper, we proposed a new algorithm based on ISQF structure, which solves the mentioned problem. Compared to the previous methods, ISQF provides improved error rates with simple hash table structure and bit operations. In fact, ISQF achieves nearly the optimal error rates. In some data flow scenarios, the error rate of the proposed algorithm is close to zero.

We also provide a basic parallel framework implementation of ISQF to meet the need of distributed applications. Experimental results show that ISQF is much better than contrast method, and ISQF exhibits near optimal error rate. In the field of repeated data detection, ISQF is a very powerful, attractive and memory efficient architecture.

Further research and empirical analysis on data flow with conceptual drift, as well as a complete architecture design and implementation of ISQF, are our future research directions.

Acknowledgements

This work is supported by National Key Research and Development Project (Grant No. 2016YFB0801204 and 2016YFB0801503).

References

[1] Alon N., Matias Y., and Szegedy M., "The Space Complexity of Approximating the Frequency Moments," *in Proceedings of 28th Annual ACM Symposium on Theory of Computing*, Philadelpia, pp. 20-29, 1996.

[2] Babcock B., Babu S., Datar M., Motwani R., and Widom J., "Models and Issues in Data Stream Systems," *in Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, Madison, pp. 1-16, 2002.

[3] Babcock B., Datar M., and Motwani R., "Load Shedding for Aggregation Queries over Data Streams," *in Proceedings of 20th International Conference on Data Engineering*, Boston, pp. 350-361, 2004.

[4] Baboescu F. and Varghese G., "Scalable Packet Classification," *in Proceedings of Applications, Technologies, Architectures, and Protocols for Computers Communications*, San Diego, pp. 199-210, 2001.

[5] Bender M., Farach-Colton M., Johnson R., Kraner R., Kuszmaul B., Medjedovic D., Montes P., Shetty P., Spillane R., and Zadok E., "Don't Thrash: How to Cache Your Hash on Ash," *in Proceedings of Ceedings of the VLDB Endoment*, Istanbul, pp. 1627-1637, 2012.

[6] Bloom B., “Space/Time Trade-offs in Hash Coding with Allowable Errors,” *Communications of the ACM*, vol. 13, no. 7, pp. 422-426, 1970.

[7] Broder A. and Mitzenmacher M., “Network Applications of Bloom Filters: A Survey,” *Internet Mathematics*, vol. 1, no. 4, pp. 485-509, 2004.

[8] Borg M., Runeson P., Johansson J., and Mäntylä M., “A Replicated Study on Duplicate Detection: Using Apache Lucene to Search Among Android Defects,” *in Proceedings of 8th International Symposium on Empirical Software Engineering and Measurement*, Torino, pp. 1-4, 2014.

[9] Chen Y., Kumar A., and Xu J., “A New Design of Bloom Filter for Packet Inspection Speedup,” *in Proceedings of 50th Annual IEEE Global Telecommunications Conference, GLOBECOM*, Washington, pp. 1-5, 2007.

[10] Chowdhury A., Frieder O., Grossman D., and McCabe M., “Collection Statistics for Fast Duplicate Document Detection,” *ACM Transactions on Information Systems*, vol. 20, no. 2, pp. 171-191, 2002.

[11] Conrad J., Guo X., and Schriber C., “Online Duplicate Document Detection: Signature Reliability in A Dynamic Retrieval Environment,” *in Proceedings of the 12th International Conference on Information and knowledge Management*, New Orleans, pp. 443-452, 2003.

[12] Cormode G. and Muthukrishnan S., “An Improved Data Stream Summary: The Count Min Sketch and Its Applications,” *Journal of Algorithms*, vol. 55, no. 1, pp. 58-75, 2005.

[13] Deng F. and Rafiei D., “Approximately Detecting Duplicates for Streaming Data Using Stable Bloom Filters,” *in Proceedings of the ACM SIGMOD International Conference on Managment of Data*, Chicago, pp. 25-36, 2006.

[14] Dharmapurikar S., Krishnamurthy P., and Taylor D., “Longest Prefix Matching Using Bloom Filters,” *IEEE/ACM Transactions on Networking*, vol. 14, no. 2, pp. 397-409, 2006.

[15] Dharmapurikar S., Krishnamurthy P., Sproull T., and Lockwood J., “Deep Packet Inspection Using Parallel Bloom Filters,” *IEEE Micro*, vol. 24, no. 1, pp. 52-61, 2004.

[16] Dutta S., Bhattacherjee S., and Narang A., “Twards “Intelligent Compression” in Streams: A Biased Reservoir Sampling Based Bloom Filter Approach,” *in Proceedings of the 15th Interntional Conference on Extending Database Technology*, Berlin, pp. 228-238, 2012.

[17] Dutta S., Narang A., and Bera S., “Streaming Quotient Filter: A Near Optimal Approximate Duplicate Detection Approach for Data Streams,” *Proceedings of the VLDB Endowment*, vol. 6, no. 8, pp. 589-600, 2013.

[18] Fan L., Cao P., Almeida J., and Broder A., “Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol,” *Computer Communiction Review*, vol. 28, no. 4, pp. 254-265, 1998.

[19] Feng W., Kandlur D., Saha D., and Shin K., “Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness,” *in Proceedings of 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, Anchorage, pp. 1520-1529, 2001.

[20] Garcia-Molina H., Ullman J., and Widom J., *Database System Implementation*, Prentice Hall, 2000.

[21] Gehrke J., Korn F., and Srivastava D., “On Computing Correlated Aggregates over Continual Data Streams,” *in Proceedings of ACM SIGMOD International Conference on Management of Dta*, Santa Barbara, pp. 13-24, 2001.

[22] Golab L., DeHaan D., Demaine E., Lpez-Ortiz A., and Munro J., “Identifying Frequent Items in Sliding Windows over On-Line Packet Streams,” *in Proceedings of the 3rd ACM SIGCOMM Coference on Internet Measurement*, Miami Beach, pp. 173-178, 2003.

[23] Gupta P. and McKeown N., “Packet Classification on Multiple Fields,” *in Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, NY, pp. 147-160, 1999.

[24] Køien G., “A Brief Survey of Nonces and Nonce Usage,” *in Proceedings of the 9th International Conference on Emerging Security Information, Systems and Technologies*, Iaria Xps Press, pp. 85-91, 2015.

[25] Kumar A., Xu J., Wang J., Spatschek O., and Li L., “Space-Code Bloom Filter for Efficient PerFlow Traffic Measurement,” *in Proceedings of IEEE INFOCOM Conference on Computer Communications 20 th Annual Joint Conference of the IEEE Computer and Communications Socities*, Hongkong, pp. 1762-1773, 2004.

[26] Lee D. and Hull J., “Duplicate Detection for Symbolically Compressed Documents,” *in Prceedings of 5th International Conference on Document Analysis and Recognition*, Banglore, pp. 305-308, 1999.

[27] Little M., Shrivastava S., and Speirs N., “Using Bloom Filters to Speed-up Name Lookup in Ditributed Systems,” *Computer Journal*, vol. 45, no. 6, pp. 645-652, 2002.

[28] Mitzenmacher M., “Compressed Bloom Filters,” *IEEE/ACM Transactions on Networking*, vol. 10, no. 5, pp. 604-612, 2002.

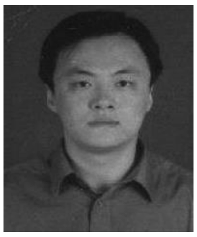
[29] Reiter M., Anupam V., and Mayer A., “Detecting Hit Shaving in Click-Through Payment

Schemes," *in Proceedings of the 3rd Conference on USENIX Workshop on Electronic Commerce*, Boston, pp. 155-166, 1998.

[30] Shen H. and Zhang Y., "Improved Approximate Detection of Duplicates for Data Streams over Sliding Windows," *Journal of Computer Science and Technology*, vol. 23, no. 6, pp. 973-987, 2008.

[31] Song H., Dharmapurikar S., Turner J., and Lockwood J., "Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing," *in Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Achitectures, and Protocols for Computer Comunication*, Philadelphia, pp. 181-192, 2005.

Shiwei Che is currently a Ph.D. candidate in the Department of Computer Science and Technology, Harbin Engineering University. He received his M.E. degree in 2010 from the Department of Computer Science and Technology of Xinjiang University, Xinjiang, China. His main research interests include social networks and community detection.



Wu Yang received a Ph.D. degree in Computer System Architecture Specialty of Computer Science and Technology School from Harbin Institute of Technology. He is currently a professor and doctoral supervisor of Harbin Engineering University. His main research interests include wireless sensor network, peer-to-peer network and information security. He is a member of ACM and senior member of CCF.

Wei Wang received a Ph.D. degree in Computer System Architecture Specialty of Computer Science and Technology School from Harbin Institute of Technology. He is currently an professor in Harbin Engineering University. His main research interests include social networks and community detection.